Introduction to Graphics

Assignment 4: Shading

Student: Kasper Passov

1 Implementation theory

A vertex program manipulates the attributes of vertices, where the fragment shader handles the pixels between verteces. This means for the color to be gradualy changing from one vertice of a triangle to another vertice, I would need to define the corner colors in the vertex program, and let the interpolation in the fragment program handle the gradient.

2 Vertix program implementation

The functions and variables part of the program needed on every vertex are handled in the vertix program, this is done by manipulation of the *uModelMatrix* and *uPerpectiveMatrix* matrices.
The implementation is supposed to create a matrix that can be defined and calculatet in the main program on the CPU, and multiplied onto the uModelMatrix using GPU cycles by letting the Vertix program handle the matrix manipulations.

3 Fragment program implementation

The fragment program will be responsible for calculation light intensity from that point into the view. This is done in the fragment program to create more beautifull lighting. The fragment program also handles colors and interpolation.

4 Vertex and Fragment variables in the main program

To define Vertex variables in the main program, it has to be added to the string called sVertexShader

```
static const std::string sVertexShader = "
                                          #version 110

                                          attribute vec3 aPosition;

                                          uniform mat4 uModelMatrix;
                                          uniform mat4 uPerspectiveMatrix;

                                          void main() {
                                                  gl_Position = uPerspectiveMatrix * uModelMat
                                          }
";
```

these variables are given values in the main program by the function glUnitform, that binds value into a variable as follows:

```
glUniformMatrix4fv(glGetUniformLocation(triangleShaderProgram.getProgram(), "uModelM
glUniform3f(glGetUniformLocation(triangleShaderProgram.getProgram(), "uColour"), 1.0
```

In the vertex program, I use the initializeBuffer function to define the starting values of my buffers, and the draw function to use these buffers.

5 C++ implementation

I did not complete the implementation as i was unsure of how to calculate the V vector. I belive I could calculate it by finding the middle of the triangle by the vertices and then minus that with the L vector.

```
static void Phong(const glm::vec3& Ia, // vec3 ambient light source(Ired, Igreen, Iblue)
                  const glm::vec3& Ip, // vec3 point intensity (Ipred, Ipgreen, Ipblue)
                  const glm::vec3& L, // light source position
                  ////////////////////  material properties
                  float Ka, const glm::vec3& Oa,
                  float Kd, const glm::vec3& Od,
                  float Ks, const glm::vec3& Os,
                  float Fatt, float n)
```

The needed parameters are the position and intesity of each point lightsource, the ambient light, and material and color of each surface.

```
glm::vec3 p1 (-33.978017f, -34.985076f,  50.214926f);
glm::vec3 p2 ( 84.192943f, -13.784394f, -50.214926f);
glm::vec3 p3 (-16.236910f,  83.754546f, -50.214926f);
glm::vec3 e1 (p1-p2);
glm::vec3 e2 (p2-p3);
glm::vec3 e3 (p3-p1);
glm::vec3 N = ((glm::cross(e1,e2) + glm::cross(e2,e3) + glm::cross(e3,e1))/glm::vec3(3,3
glm::vec3 R = N * glm::vec3(2,2,2) * (N * L) - L;
```

I define each vertice and edge to find N and R.

```
glm::vec3 fattv (Fatt, Fatt, Fatt);
float IPhong = Ka * Od * Ia + //Ambient
               Kd * Od * fattv * Ip * (L * N) + //Diffuse
               Ks * Os * fattv * Ip * glm::pow((R * V),n); // specular
```

Then i do the calculations and save the matrix in a uIPhong variable for the fragment and vertex program to use.